Push relabel maximum flow algorithm

Seminarski rad u okviru kursa
Konstrukcija i analiza algoritama 2
Matematički fakultet

Anđela Damnjanović
mi19059@alas.matf.bg.ac.rs

19. maj 2024.

Sažetak

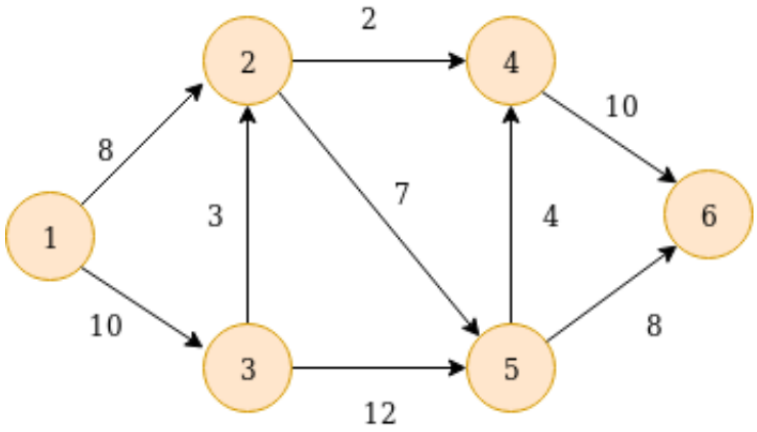
Teorija grafova i grafovski problemi imaju veoma važno mesto u matematičkoj teoriji, ali ujedno nalaze i brojne praktične primene sa kojima se ljudski rod svakodnevno susreće — od navigacije pa sve do generisanja rasporeda časova ili modela atoma i molekula. Upravo zbog njihove velike rasprostranjenosti, razvijeni su mnogi algoritmi koji imaju za cilj da što efikasnije reše pomenute probleme. U skladu sa tim, centralna tema ovog rada je upravo opis jednog veoma bitnog algoritma koji služi za optimizaciju transportne mreže, a koji je poznat pod nazivom Push-relabel maximum flow algorithm.

Sadržaj

1 Uvod

Grafovi su matematičke strukture koje se sastoje od skupa *čvorova* (koji se, po konvenciji, obeležava slovom V) i skupa *grana* (koji se obeležava slovom E). Oni predstavljaju pogodnu strukturu podataka za modelovanje najrazličitijih problema na koje ljudi nailaze. Mogu se koristiti za pravljenje modela molekula u hemiji, za modelovanje povezanosti korisnika na društvenim mrežama, za vizualizaciju povezanosti mesta na geografskoj karti i tako dalje. Stoga ni ne treba da čudi što su svoju primenu našli i u računarstvu — grafovi imaju veoma širok spektar primena poput klasterovanja, istraživanja podataka, segmentacije slika [5]. Naravno, ovo su samo neke oblasti računarstva gde su grafovi korisni.

Za sve probleme koji se mogu predstaviti grafovima, definisani su odgovarajući algoritmi koji ih rešavaju. Međutim, ne mogu se svi algoritmi primeniti na sve grafove. Na primer, nalaženje minimalnog razapinjućeg stabla nije moguće primeniti na usmerene grafove, a nalaženje Ojlerovog puta neće biti moguće ako je broj čvorova sa neparnim stepenom različit od 0 i 2 [2]. Analogno, ni push-relabel maximum flow algoritam, koji je centralna tema ovog rada, ne može se primeniti na svaki graf. Da bi pomenuti algoritam bio primenljiv, graf mora biti *transportna mreža*, tj. neophodno je da graf bude povezan, usmeren, težinski i sadrži tačno jedan čvor sa ulaznim stepenom 0 iz koga će se propuštati tok (stoga prikladno nazvan *izvor*) i jedan čvor sa izlaznim stepenom 0 u koji će sav tok uvirati (te je nazvan *ponor*)[1]. Takođe, neophodno je i da grane grafa budu pozitivne težine. Svaka ovakva težina označava najveću količinu toka koju je moguće propustiti kroz granu i naziva se *kapacitet* grane. Primer jedne transportne mreže prikazan je na slici 1.



Slika 1: Grafički prikaz transportne mreže [3]

[1]Ukoliko u grafu postoji više izvora i ponora, potrebno je dodati nova dva čvora: jedan od koga će grane voditi ka svakom izvoru i jedan do koga će grane voditi od svakog ponora, čime se problem svodi na već poznati problem.

Osobine toka koje je potrebno zadovoljiti su:

1. Količina toka koja se propušta kroz granu ne sme biti manja od 0 i ne sme prevazilaziti kapacitet te grane
2. Ukupna količina toka koja uđe u čvor mora biti jednaka količini toka koja taj isti čvor napušta [2].

Kao što se iz naziva može zaključiti, transportne mreže su pogodne za modelovanje fizičkog transporta sa jednog mesta na drugo, npr. mogu se koristiti za modelovanje cevovoda ili pak saobraćaja. Međutim, transportne mreže mogu naći svoju primenu i za modelovanje rasporeda sedenja ili rasporeda časova, za višeciljne optimizacije, kao i za segmentaciju slika [6, 3].

U nastavku će biti opisani dosadašnji rezultati poznatih matematičara u poglavlju 3, biće izveden odgovarajući algoritam u poglavlju **??** koji će biti testiran na različitim ulazima u delu 5, te će se na osnovu njih donostiti zaključak u poglavlju 6.

2 Opis problema i dosadašnji rezultati

Nakon upoznavanja sa osnovnim pojmovima koji su potrebni za razumevanje algoritma, moguće je posvetiti pažnju detaljnom opisu problema. Neka je data transportna mreža. Potrebno je pronaći maksimalni tok kroz nju, tj. potrebno je odrediti koja je najveća količina supstance koja se može pustiti iz izvora i, putujući granama zadatog grafa prema pravilima opisanim u poglavlju 1, stići do ponora.

Jedan od prvih algoritama koji rešavaju dati problem osmišljen je davne 1956.godine i nosi naziv Ford[3]-Fulkersonov[4] algoritam. On pripada grupi pohlepnih algoritama i zasniva se na *povećavajućim putevima*. Sam algoritam je veoma jednostavan: potrebno je najpre pronaći put od izvora do ponora i pamtiti koji je najmanji kapacitet grana koje se nalaze na tom putu. Zatim se od izvora ka ponoru propušta odgovarajuća količina supstance (jednaka upravo pronađenom minimalnom kapacitetu) i dati postupak se ponavlja dokle god postoji put između izvora i ponora.

Dakle, algoritam se završava onda kada više nema povećavajućih puteva. Međutim, ne postoji garancija da će se algoritam uopšte završiti, pa on samim tim garantuje ispravan rezultat samo u slučaju kada se algoritam može završiti u konačno mnogo koraka. Završetak algoritma garantuje se u slučaju da su težine grana racionalni brojevi. Složenost datog algoritma (u slučajevima kada se on zaustavlja) je $O(Ef)$ gde je f vrednost maksimalnog toka kroz graf, jer je za pronalaženje puta od izvora do ponora potrebno obići sve grane grafa, dok je broj uvećavajućih puteva koje treba pronaći između 1 i f.

Kako bi se rešio problem terminacije Ford-Fulkersonovog algoritma, autori Edmonds[5] i Karp[6] su nezavisno došli do načina da praktično implementiraju Ford-Fulkersonov algoritam tako da se garantuje zaustavljenje algoritma, te je ova implementacija nazvana po njima. Jedina izmena

[2] Ova osobina mora važiti za sve čvorove osim za izvor i ponor.

[3] Lester Rendolf Ford (1927-2017) bio je američki matematičar koji se specijalizovao upravo za probleme transportnih mreža.

[4] Delbert Rej Fulkerson (1924-1976) takođe je bio američki matematičar.

[5] Džek Edmonds (1934-) je američki informatičar i matematičar, najpoznatiji po svom doprinosu u poljima optimizacije, kombinatorike i diskretne matematike.

[6] Ričard Maning Karp (1935-) je američki informatičar, najpoznatiji po doprinosima u oblasti teorije algoritama. Takođe, jedan je od dobitnika prestižne Tjuringove nagrade.

u odnosu na originalni algoritam leži u tome što se za uvećavajući put bira *najkraći* takav put u grafu. Složenost datog algoritma je O(E^2 V). Ponovo, vreme da se nađe uvećavajući put zahteva O(E) vremena, dok je najveći broj povećavanja puteva O(EV).

Kao što se može videti, pomenuti algoritmi nisu toliko efikasni ili čak nisu uvek terminirajući. Stoga je razvijen još jedan algoritam, koji je i efikasniji od Edmonds-Karpovog, ali se i uvek garantuje njegovo zaustavljanje za razliku od generičnog Ford-Fulkersonovog algoritma. Taj algoritam dobio je naziv Push-relabel maximum flow algoritam i zasniva se na propuštanju određene količine toka iz jednog čvora u njemu susedan (ova operacija, koja se čak našla i u samom imenu algoritma, nazvana je push) ukoliko je to moguće i u ponovnom obeležavanju čvorova (operacija relabel) ukoliko to nije moguće. Jedna od osnovnih razlika u odnosu na prethodne pristupe podrazumeva postojanje takozvanog *predtoka* (eng. *preflow*) koji zasićuje sve grane koje izlaze iz izvora. Još jedna novina u odnosu na već postojeće algoritme jeste obeležavanje čvorova. Ideja je da se tok može propustiti iz jednog čvora u drugi pod uslovom da se prvi nalazi na većoj "visini"od drugog. Dakle, visinama je implicitno zadata hijerarhija među čvorovima, gde čvorovi koji se nalaze na višem nivou hijerarhije mogu da propuste supstancu ka čvorovima sa nižih nivoa, ali ne i obrnuto. Stoga je za svaki čvor bitno čuvati njegovu trenutnu visinu [1].

3 Opis algoritma

Algorithm 1 Generic push-relabel maximum flow algorithm

Require: graph with one source node, one sink node and edges with positive capacities

```
    Initialization:
        makeResidualGraph
        initializePreflow
while ∃ an active node do
    while excessFlow > 0 do
        if can push from active node then
            push flow to a neighbouring node
        else if !can push then
            relabel active node
        end if
    end while
        remove node from the list of active nodes
end while
```

Najpre je potrebno uvesti pojam *rezidualnog grafa*. To je graf koji se dobija nakon propuštanja toka kroz originalni graf i sastoji se od svih grana koje ulaze u sastav originalnog grafa s tim što se vrednost toka grana iz originalnog grafa smanjuje za vrednost toka koji je propušten kroz njih, nakon čega se dodaju grane suprotnog usmerenja granama iz originalnog grafa kroz koje je propušten tok, a vrednost njihovog toka se uvećava (ili inicijalizuje ako data grana već ne postoji) za vrednost koja je kroz odgovarajuće grane propuštena.

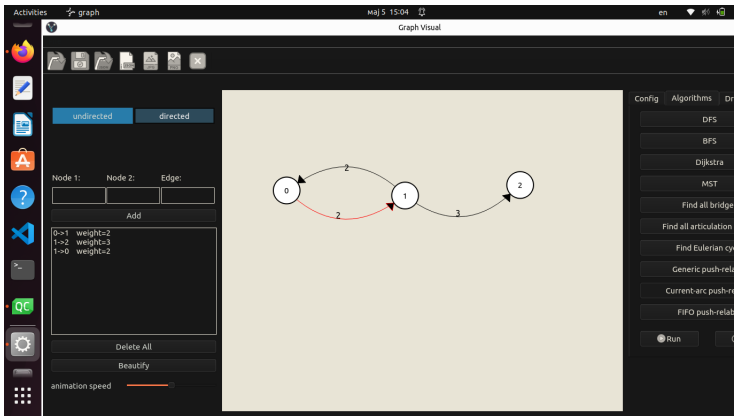
Dakle, u prvom koraku algoritma u rezidualni graf biće dodate grane koje vode od suseda izvornog čvora do njega samog. Nakon toga, potrebno je inicijalizovati visine čvorova. Pošto je najpre potrebno propustiti tok iz izvornog čvora, on intuitivno treba da ima najveću visinu. Stoga njegovu visinu postavljamo na vrednost ukupnog broja čvorova. Svi ostali čvorovi imaju visinu 0. Zatim je potrebno napraviti odgovarajući *predtok*, tj. od izvora do njegovih suseda propustiti maksimalni mogući tok (to je količina toka koja odgovara kapacitetu grane). Istovremeno, vrednosti *excessFlow* promenljive kod čvorova koji su susedni izvoru postavljaju se na vrednosti kapaciteta grane koja vodi od izvora do njih, dok se vrednost iste promenljive u izvoru smanjuje za odgovarajuću vrednost. Nakon što je tok propušten kroz granu originalnog grafa, potrebno je vrednost toka odgovarajuće rezidualne grane umanjiti za vrednost kapaciteta grane. Ovim je završen proces inicijalizacije.

Svi čvorovi koji imaju višak toka u sebi moraju ga proslediti do ponora (ili deo vratiti nazad u izvor ukoliko ne postoji način da se sav tok propusti do ponora). Dakle, unutrašnji čvorovi grafa ne smeju u sebi imati viška toka kada se algoritam završi. Zato se oni proglašavaju *aktivnim* čvorovima, tj. čvorovima koji mogu da propuste tok. Nakon inicijalnog koraka jedini aktivni čvorovi su neposredni susedi izvornog čvora, što se može promeniti tokom izvršavanja algoritma. Zato je potrebno obići čvorove u svakoj iteraciji.

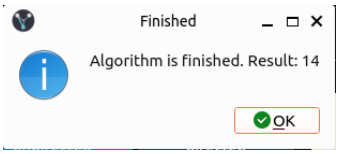
Ako postoji aktivan čvor, potrebno ga je obraditi. Ukoliko je visina nađenog čvora veća od visine nekog od njegovih suseda i ako je tok kroz granu koja ih povezuje manji od kapaciteta te grane, moguće je propustiti tok iz aktivnog čvora u njegovog suseda. Količina toka koja će biti propuštena je *delta = min(u->excessFlow, e->capacity - e->flow)*, gde je *u* trenutni aktivan čvor, a *e* grana koja vodi od *u* do njegovog suseda. Nakon propuštanja toka iz aktivnog čvora, potrebno je napraviti odgovarajuću rezidualnu granu koja će imati isti kapacitet kao i grana originalnog grafa. Zatim se povećava vrednost toka grane kroz koju je propušten tok, dok se vrednost toka napravljene rezidualne grane smanjuje za istu vrednost.

Sa druge strane, ukoliko nije moguće propustiti tok kroz aktivni čvor, potrebno je dati čvor ponovo obeležiti. Obeležavanje podrazumeva promenu visine aktivnog čvora. Na osnovu samog opisa algoritma, do ove situacije je moguće doći ukoliko ne postoji čvor koji je sused aktivnom čvoru a koji ima manju visinu od njega. Dakle, susedni čvorovi imaju ili veće ili jednake visine visini trenutnog aktivnog čvora. Kako je za propuštanje toka iz čvora neophodno da on bude na većoj visini od čvora ka kome je moguće propustiti odgovarajuću vrednost toka, intuitivno je potrebno podići aktivni čvor na visinu iznad nekog od njegovih suseda kako bi bilo moguće propustiti tok. Visina koja omogućava da se push operacija može izvesti jeste minimalna visina njegovih suseda uvećana za 1. Dalje algoritam teče na isti način sve dok svi čvorovi osim izvora i ponora imaju višak toka u sebi. Vrednost maksimalnog toka može se pročitati iz vrednosti *excessFlow* promenljive čvora koji je ponor. Na slikama 2 i 3 može se videti način vizualizacije push operacije i dobijanja krajnjeg rezultata.

Iako je ovaj algoritam efikasniji od Edmonds-Karpovog, postoje načini da se i njegova efikasnost unapredi. Glavnu prepreku u ostvarivanju efikasnosti predstavlja ponovni obilazak svih čvorova u svakoj iteraciji prilikom potrage za aktivnim čvorom. Ispostavlja se da nije potrebno uvek obilaziti sve čvorove, već je moguće čuvati listu aktivnih čvorova i nju ažurirati. Pristup listi vrši se u amortizovanom vremenu O(1), što doprinosi smanjenom vremenu izvršavanja algoritma. Na samom početku, lista sadrži samo



Slika 2: Grafički prikaz izvršavanja jedne push operacije



Slika 3: Grafički prikaz rezultata izvršavanja algoritma

susede izvornog čvora, dok se kasnije, nakon izvršene *push* operacije u nju dodaju čvorovi koji su primili deo toka. Kada se iz aktivnog čvora više ne može propustiti tok ni u jedan od njegovih suseda, potrebno ga je ponovo obeležiti. Kada u njemu nema više viška toka, on se briše iz liste aktivnih. Složenost ovako optimizovanog algoritma je $O(V^3)$, jer se aktivan čvor biti izabran $O(V)$ puta, a složenost *relabel* operacije je $O(V^2)$.

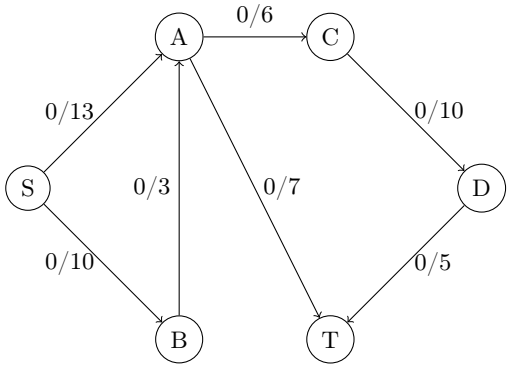
Takođe treba napomenuti da postoje i različiti načini da se odabere aktivni čvor. Pametnim izborom aktivnog čvora može se dodatno smanjiti vremenska složenost algoritma. Postoje sledeći načini selekcije:

- *FIFO selekciono pravilo* podrazumeva da se aktivni čvorovi čuvaju u redu. Nakon svakog propuštanja toka element sa početka reda se uklanja, dok se u red dodaje čvor koji je sada postao aktivan. Složenost ovog algoritma je $O(V^3)$.
- *Obeleženi na početak selekciono pravilo* podrazumeva da se čvor koji je ponovo obeležen *relabel* operacijom bude pomeren na početak liste aktivnih čvorova, te da se iz njega dalje propušta tok. Složenost ovog algoritma je takođe $O(V^3)$.
- *Selekciono pravilo najveće oznake* podrazumeva da se u svakom koraku od svih aktivnih čvorova bira onaj koji ima najveću visinu. Složenost ovog algoritma je takođe $O(V^2\sqrt{E})$.

Za potrebe ovog rada implementirani su generički push-relabel algoritam, FIFO i obeleženi na početak selekciono pravilo.

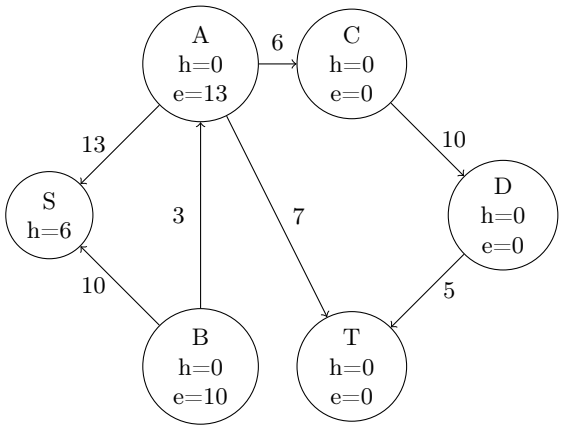
4 Primer

U cilju lakšeg razumevanja osnovne verzije algoritma (generički agoritam) u nastavku je dat primer koji slikovito objašnjava korake pri izvršavanju. Razmotrimo sledeći graf [4]. Izvorni čvor obeležen je sa S, dok je ponor obeležen sa T, dok se pored svake grane nalaze informacije o količini toka koja je kroz nju propuštena i ukupnom toku koji je moguće propustiti kroz tu granu. Na primer, oznaka $0/13$ iznad grane koja povezuje S i A označava da kroz pomenutu granu nije propušten tok, a da je kroz nju moguće propustiti najviše 13 jedinica toka.

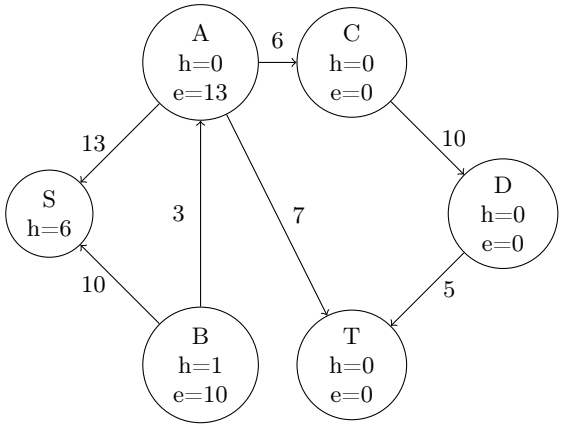


Prvi korak je, kao što je pomenuto u poglavlju **??**, inicijalizacija, tj. potrebno je napraviti rezidualni graf, inicijalizovati visine i propustiti maksimalan tok kroz sve grane koje izlaze iz izvornog čvora. Dakle, pošto je iz izvora potrebno istočiti određenu količinu supstance u njemu susedne čvorove, potrebno je dodati rezidualne grane koje će biti usmerene od suseda izvornog čvora ka samom izvoru. Maksimalna vrednost toka koji je kroz svaku ovakvu granu moguće propustiti odgovara maksimalnoj vrednosti toka koji je moguće propustiti od izvora do odgovarajućeg suseda. Neka grana između S i A opet posluži kao primer. Njoj rezidualna grana povezivaće iste čvorove, ali će biti usmerena od A ka S i maksimalna vrednost toka koji je kroz nju moguće propustiti takođe je 13, kao i u slučaju originalne grane.

Zatim je potrebno inicijalizovati visine čvorova. Dakle, potrebno je visinu izvornog čvora postaviti na vrednost koja odgovara ukupnom broju čvorova grafa. U navedenom primeru, to je 6. Sve ostale visine bivaju inicijalizovane na 0. Sada je ostalo još samo propustiti maksimalni mogući tok kroz sve grane koje ističu iz izvornog čvora. Tom prilikom, potrebno je ažurirati vrednosti toka koji je propušten kroz mrežu i vrednosti koje odgovaraju višku supstance u svakom čvoru. Radi lakše čitljivosti, grane u nastavku neće biti obeležavane u gorenavedenom obliku, već će za svaku granu biti prikazano koliko je još jedinica toka moguće propustiti kroz nju. Samim tim, ukoliko je kroz neku granu propuštena maksimalna količina toka, ona se briše iz grafa. Ovim pristupom ne dolazi do gubljenja informacija upravo zato što ako je došlo do protoka supstance iz jednog čvora u drugi sigurno postoji rezidualna grana na osnovu koje se može zaključiti kolika je bila vrednost maksimalnog toka originalne grane. Izgled grafa posle prvog koraka može se videti ispod. Oznake h i e koje se mogu videti unutar čvora predstavljaju vrednosti visine i viška supstance u čvoru, respektivno.

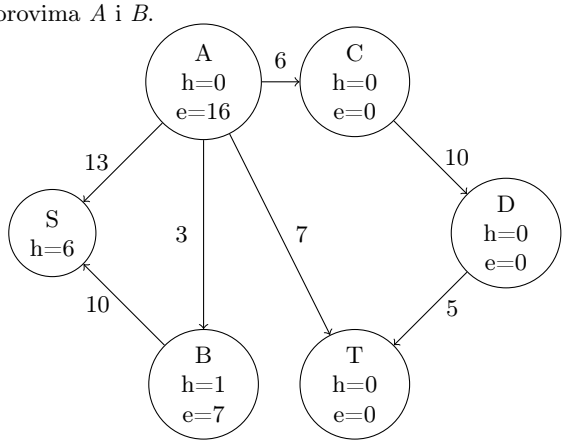


Zatim, potrebno je naći čvorove koji imaju višak supstance u sebi da bi se tok mogao dovesti do svoje krajnje destinacije, tj. do ponora. Sa gornjeg grafika može se videti da su to čvorovi A i B. Neka je algoritam prvo naišao na čvor B. Potrebno je najpre proveriti da li se iz datog čvora tok može propustiti do nekog od njegovih suseda. Pošto su svi susedi čvora B na većoj ili jednakoj visini od njega, propuštanje supstance iz čvora B nije moguće, već je potrebno ponovo obeležiti dati čvor. Podsećanja radi, potrebno je naći minimalnu vrednost visine njegovih suseda, uvećati je za 1 (da bi bilo moguće propustiti tok na manje visine) i dobijenu vrednost postaviti kao novu visinu čvora B. Jedini susedi koje čvor B ima su A i S. Minimum njihovih visina je 0, pa je tu vrednost potrebno inkrementirati za 1, te je nova visina čvora B jednaka 1, što se može videti na grafiku ispod.

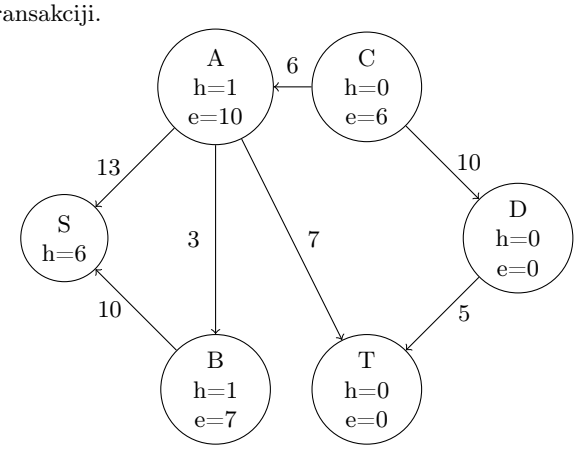


Sada je moguće propustiti tok iz B ka A. Međutim, potrebno je izračunati koliko je jedinica toka moguće propustiti. Kao što je poznato, svaki čvor čuva vrednost koja pokazuje koliki je višak supstance koji se u njemu nalazi. Idealno bi bilo propustiti sav višak odjednom, ali to nije uvek moguće. Naime, može se desiti da je višak u čvoru veći od maksimalne vrednosti toka koji se može propustiti kroz jednu granu. U tom slučaju, logično, nije moguće odjednom propustiti ceo višak iz čvora. Takođe, moguće je doći i u situaciju da je nemoguće propustiti svu količinu supstance iz čvora iako je maksimalna dozvoljena količina toka veća od viška u tom čvoru. Ova situacija se javlja kada je iz datog čvora kroz željenu granu već propuštena neka količina toka u prošlosti, pa bi u slučaju propuštanja celokupnog viška iz čvora došlo do "prekoračenja"maksimalne vrednosti

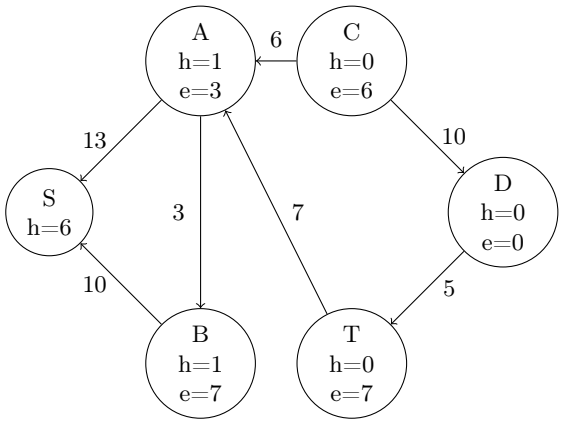
toka koju je moguće propustiti kroz datu granu. Dakle, količina toka koja se propušta mora se pažljivo izabrati. Instinktivno, rešenje je da uporede višak u čvoru i količina koju je još uvek moguće propustiti kroz granu i propustiti minimalnu od te dve veličine. Pošto je iz B u A moguće direktnom granom propustiti 3 jedinice toka, a višak u čvoru B iznosi 10 jedinica toka, kroz pomenutu granu propušta se minimum ta dva broja, to jest 3. Na kraju, potrebno je ažurirati graf i vrednosti grana i viška u čvorovima A i B.



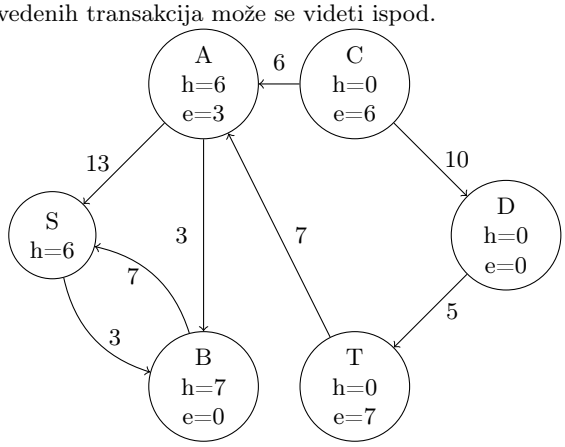
Algoritam nastavlja sa radom kao i do sad. Najpre nalazi čvor koji u sebi ima viška i može da ga propusti negde. Čvor B ima u sebi višak supstance, ali nema izlaznih grana, pa ga algoritam neće prijaviti. Sa druge strane, čvor A će biti pronađen. No, pošto su svi njegovi susedi na visinama većim ili jednakim od njega, potrebno ga je ponovo obeležiti. Kao i do sad, nalazi se minimum visina svih njegovih suseda, a to je 0 (visina čvora C), uvećava se za 1 i to se postavlje kao nova visina čvora A. Sada je moguće propustiti tok iz A u C. Analognim postupkom kao do sad, količina koju je potrebno propustiti je 6. Naravno, potrebno je ponovo ažurirati graf, njegove grane i vrednosti viška u čvorovima koji su učestvovali u transakciji.



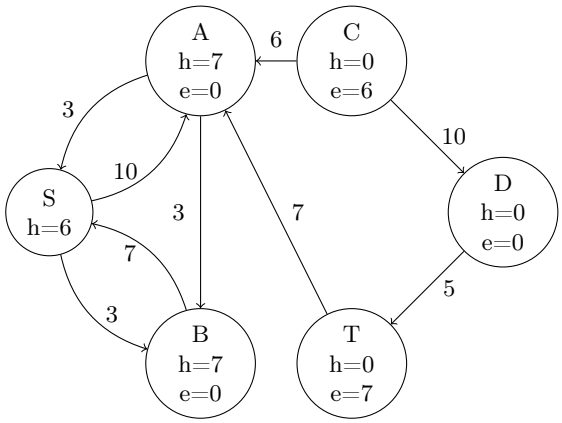
Sledeća iteracija algoritma opet vraća A kao čvor iz kog se može propustiti tok. Ovog puta, A se već nalazi na većoj visini od ponora, pa je moguće direktno poslati tok iz A u T. Vrednost toka koja se propušta je 7, koji predstavlja minimum gore opisanih vrednosti.



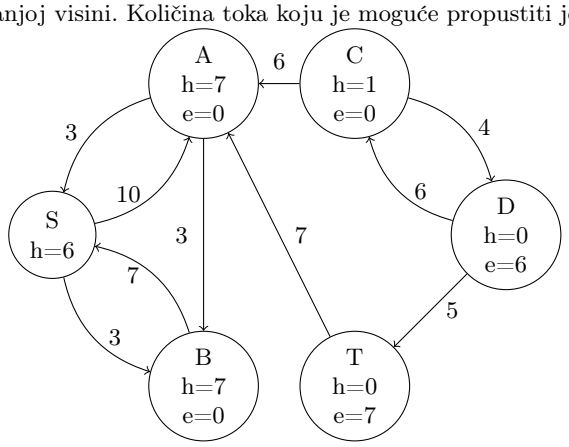
Nakon ove iteracije, algoritam ponovo pronalazi A kao aktivni čvor. No, kako je njegova visina manja ili jednaka visinama njegovih suseda, potrebno ga je preobeležiti. Ponovo je potrebno naći minimum, što je 1 (jedini susedi čvora A su B i izvor, pa je minimum njihovih visina 1). Dakle, nova visina čvor aA je 2. Sada je moguće pustiti tok iz A ka B. Vrednost tog toka je 3. Graf, grane i vrednosti viškova u čvorovima se ponovo ažuriraju na odgovraajući način. Algoritam nastavlja da radi kao i do sad, pa nailazi na B kao na sledeći aktivni čvor i, iz istih razloga kao i do sad, ponovo ga obeležava. Njegova visina se postavlja na 3 (1 više od visine čvora A, koji mu je opet postao sused) i moguće je propustiti tok iz B u A. Lako se da zaključiti da će se ovaj proces slanja toka iz A u B i nazad ponavljeti sve dok neki od njih ne dostigne visinu koja je veća od visine izvora. Prvi čvor čija će visina prestići visinu izvora je B, pa će on vratiti 7 jedinica supstance u izvor. Izgled rezultujućeg grafa nakon svih navedenih transakcija može se videti ispod.



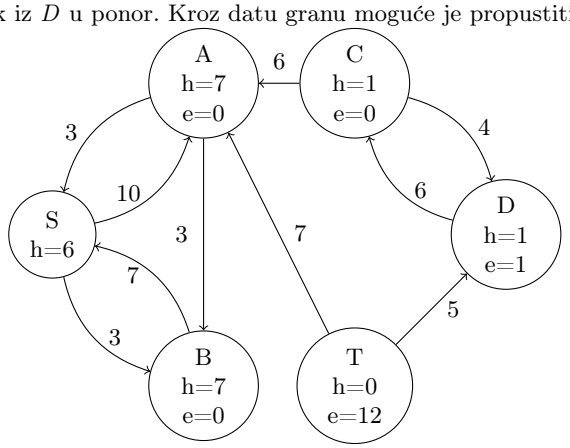
U sledećem koraku, algoritam ponovo vraća A kao aktivan čvor. Međutim, kako je njegova visina ponovo manja ili jednaka visinama njegovih suseda, potrebno ga je ponovo obeležiti. Nova visina čvora A je 7. Sada je moguće propustiti višak toka iz A u izvorni čvor, pa se 3 jednince toka vraćaju u S.



Nakon eliminacije viška toka iz A i B, algoritam vraća C kao sledeći aktivni čvor. Međutim, opet je potrebno preobeležiti visinu čvora jer ona nije veća ni od jedne visine njegovih suseda. Pošto je visina čvora C promenjena, moguće je direktno propustiti tok iz njega u suseda D koji je na manjoj visini. Količina toka koju je moguće propustiti je 6.

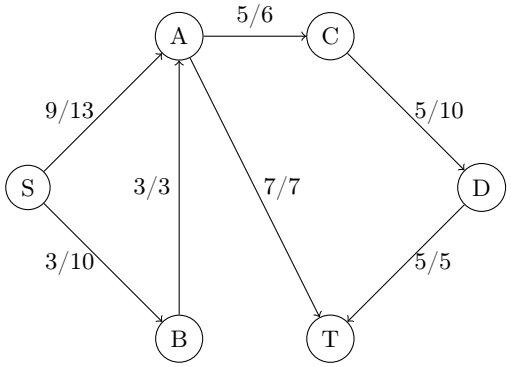


Algoritam analogno nastavlja potragu za aktivnim čvorom i pronalazi D. No, kako je njegova visina manja ili jednaka visini njegovih suseda, potrebno ga je ponovo obeležiti. Nova visina čvora D je 1 (za 1 uvećana vrednost minimalne visine susednih čvorova). Sada je moguće propustiti tok iz D u ponor. Kroz datu granu moguće je propustiti 5 jedinica toka.



Pošto je u čvoru D ostalo viška toka, on je ponovo aktivan čvor.

Međutim, potrebno ga je ponovo obeležiti jer je njegova visina jednaka visini susednog čvora. Dakle, visina čvora D postaje 2 i nakon toga je moguće vratiti višak toka u C. Nakon ažuriranja grafa, algoritam sada vraća C kao aktivan čvor. Ali, iz istih razloga kao i do sad, i njega je potrebno ponovo obeležiti, te čvor C dobija visinu 3 i prebacuje svoj višak toka u D. Ovo se ponavlja sve dok C ne prestigne visinu čvora A da bi u njega vratio svoj višak toka. A zatim samo prosleđuje ovaj višak u izvor i konačan rezultat algoritma može se videti na grafiku ispod. Dakle, maksimalan tok je 12 i rezultat se može pročitati iz vrednosti viška u čvoru C.



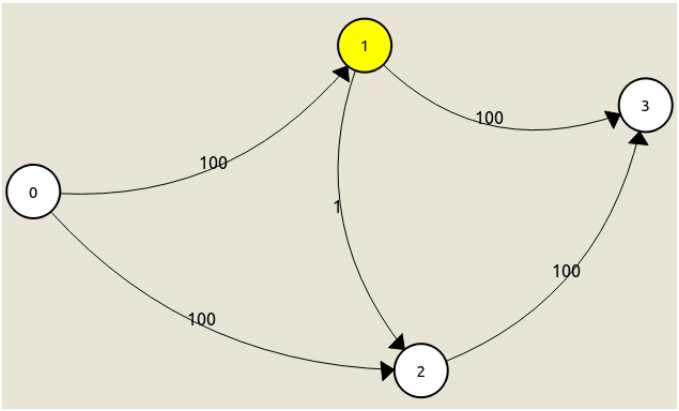
5 Implementacioni detalji i eksperimentalni rezultati

Za implementaciju pomenutog algoritma korišćeni su programski jezik C++ i QtCreator. Primer vizualizacije jedne transportne mreže može se naći na slikama 4, 5 i 6. Čvorovi i grane grafa, kao i sam graf predstavljeni su odgovarajućim klasama. Od značajnih atributa klase Node izdvajaju se polje *excessFlow* koje govori koliko "viška"toka se nalazi u datom čvoru i polje *height* koje govori koja je visina datog čvora. Sa druge strane, klasi Edge dodat je atribut *flow* koji govori koja količina toka je propuštena kroz datu granu [7].

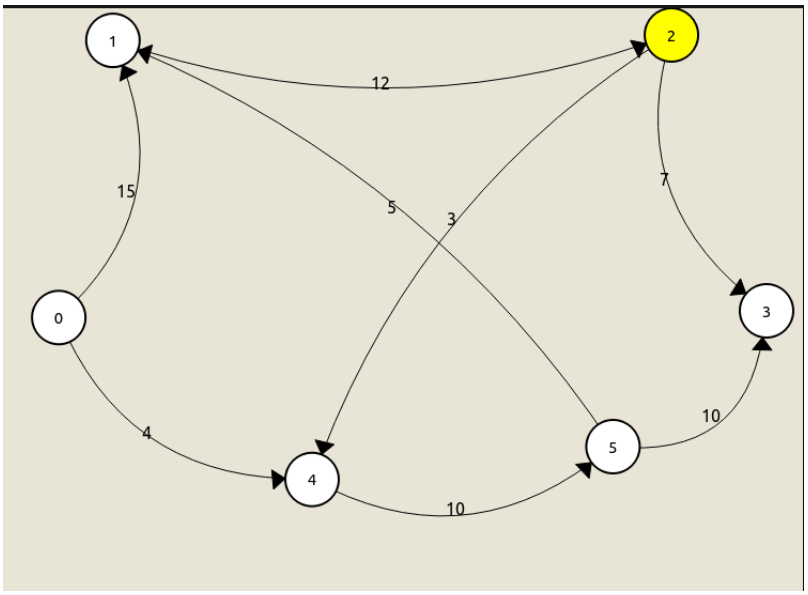
Nakon upoznavanja sa algoritmima, ovo poglavlje biće posvećeno testiranju programa i analizi dobijenih rezultata. Najpre su algoritmi pokrenuti nad transportnim mrežama koje su dostupne na internetu radi provere njihove valjanosti. Radi lakšeg snalaženja, ovi test primeri nazvani su NodesNEdgesM.json, gde su N i M redom broj čvorova i grana mreže razmatrane u odgovarajućem test primeru. Svi algoritmi dobro su se pokazali, tj. davali su korektne rezultate. No, kao što se može videti iz naziva odgovarajućih JSON fajlova, svi pronađeni test primeri sastoje od malog broja čvorova i grana, te je za detaljniju analizu potrebno je generisati masivnije grafove.

Zarad detaljnijeg testiranja, napisan je program generateTests.ipynb čija je uloga da generiše usmerene grafove sa željenim brojem čvorova i nasumično odabranim granama. Zatim je potrebno od datog grafa napraviti transportnu mrežu procesom koji je opisan u uvodnom poglavlju. Na kraju, potrebno je generisane grafove smestiti u odgovarajuće .json fajlove kako bi ih bilo lakše učitati i testirati algoritme na njima. Generisani su

[7] Polje koje čuva težinu grane (kapacitet u slučaju push-relabel algoritma) već postoji u implementaciji.



Slika 4: Prikaz transportne mreže sa 4 čvora

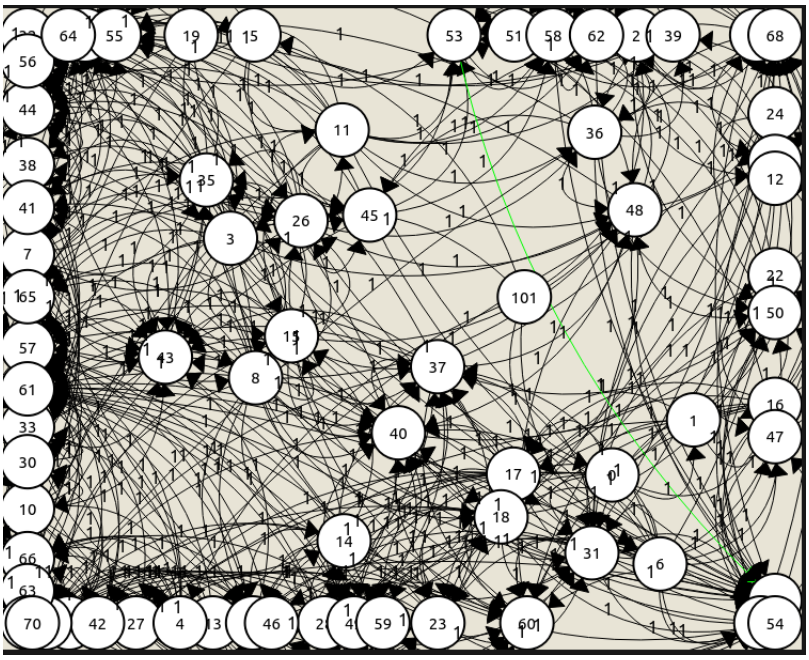


Slika 5: Prikaz transportne mreže sa 6 čvorova

grafovi sa 20, 30, 40, 50, 60, 70, 80, 90 i 100 čvorova, a ovi test primeri nazvani su NodesNedgesM.json, gde N označava broj čvorova od kojih se test primer sastoji, a M broj grana u grafu.

Rezultati izvršavanja algoritama mogu se naći u tabeli 5, a poređenje vremena izvršavanja na slici 7. Kao što se iz priloženog može videti, i FIFO strukture i u praksi su se pokazale dosta bolje od generičkog algoritma, pogotovo za veće ulazne vrednosti. Kako su oba efikasna algoritma iste složenosti, ne treba da čudi što su i vremena izvršavanja prilično ujednačena, s tim što FIFO struktura ipak pokazuje za nijansu bolje performanse u svim navedenim primerima.

Pored veličine ulaza, bilo bi zanimljivo proveriti da li još neki atribut grafa utiče na vreme izvršavanja algoritama. Stoga je napravljen novi set

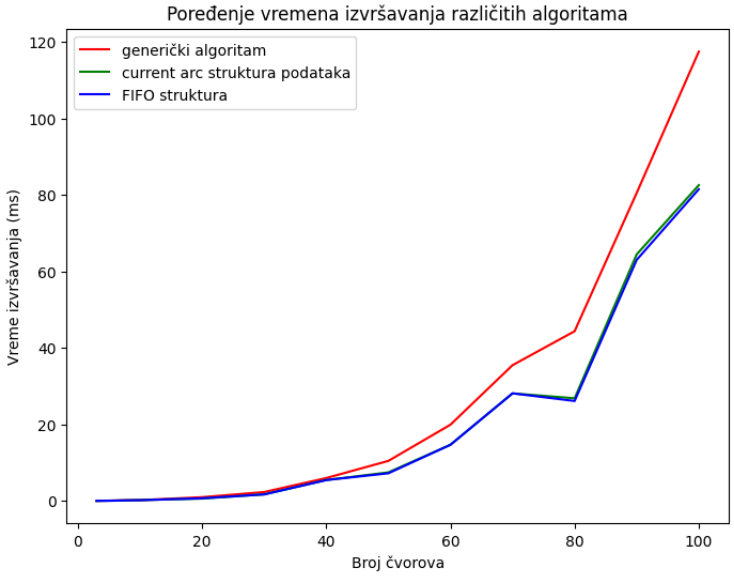


Slika 6: Prikaz transportne mreže sa 80 čvorova

Tabela 1: Pregled vremena izvršavanja algoritama

V	Generički (ms)	obeleženi na početak (ms)	FIFO (ms)
3	0.0496	0.0251	0.0210
4	0.05687	0.054	0.0469
6	0.1123	0.092	0.0889
8	0.1995	0.152	0.163
10	0.2255	0.222	0.2029
20	0.9851	0.624	0.696
30	2.328	1.723	1.712
40	5.9887	5.474	5.464
50	10.466	7.466	7.226
60	19.9287	14.686	14.67
70	35.4815	28.11	28.12
80	44.3591	26.83	26.59
90	80.507	64.44	63.01
100	117.419	82.525	81.47

test primera takvih da svi grafovi imaju po 50 čvorova, dok se njihove *gustine* – odnos broja grana koje se nalaze u grafu i ukupnog mogućeg broja grana – razlikuju. Dobijeni rezultati su tabelirani i mogu se videti u tabeli 5, a grafik zavisnosti brzine izvršavanja algoritama od gustine grafa prikazan je na slici 8. Kao što se iz priloženog može videti, algoritmi koji koriste FIFO i obeleženi na početak strukturu i dalje beleže bolje rezultate od generičkog algoritma (ponovo FIFO struktura daje za nijansu bolje



Slika 7: Poređenje vremena izvršavanja algoritama

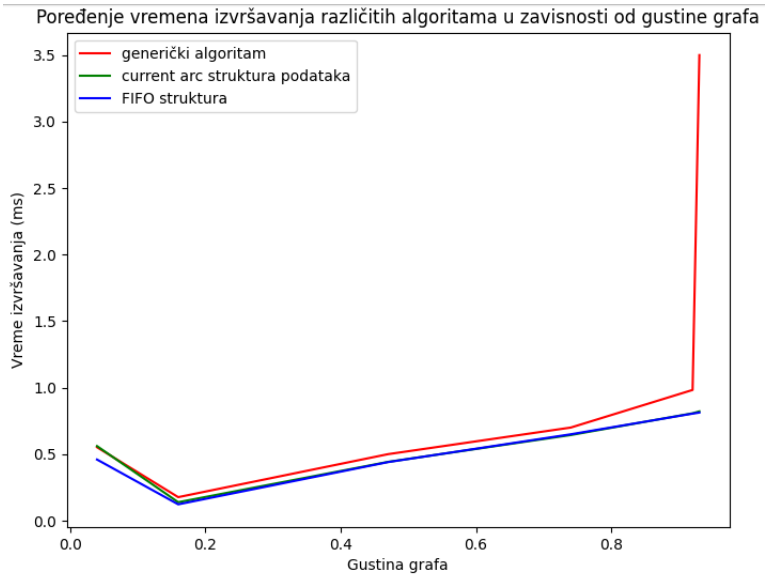
rezultate). Dalje, iz dobijenih rezultata može se izvući zaključak da vreme izvršavanja algoritama zavisi i od gustine grafa. Naime, dužina trajanja algoritama pokrenutih nad ovim grafovima se drastično razlikuje iako svi oni imaju isti broj čvorova. Takođe, zanimljivo je primetiti da se u nekim slučajevima algoritmi pokrenuti nad gušćim grafovima završavaju brže nego oni pokrenuti nad grafovima manje gustine (npr. svi algoritmi su efikasniji nad grafovima gustine 0.36 i 0.47 nego nad grafovima gustine 0.16 ili 0.25). Dakle, ne važi nužno činjenica da sa porastom gustine raste i vreme izvršavanja. To može dovesti do zaključka da nije gustina jedini faktor koji utiče na dužinu izvršavanja algoritma, već ona verovatno zavisi i od unutrašnje strukture grafa.

Tabela 2: Zavisnost vremena izvršavanja algoritama od gustine

Gustina grafa	**Generički (ms)**	**obeleženi na početak (ms)**	**FIFO (ms)**
0.04	0.176	0.138	0.122
0.16	0.552	0.560	0.459
0.25	0.599	0.356	0.208
0.36	0.528	0.332	0.330
0.47	0.534	0.442	0.440
0.74	0.699	0.643	0.649
0.92	0.988	0.807	0.804

6 Zaključak

Algoritmi koji su bili izloženi pokazali su se dobro u poređenju sa jednostavnim Edmonds-Karpovim algoritmom. Takođe, za razliku od Ford-



Slika 8: Poređenje vremena izvršavanja algoritama u odnosu na gustinu grafa

Fulkersonovog algoritma, svaki od navedenih algoritama se garantovano završava nezavisno od ulaznih vrednosti. Očekivano, testovi su pokazali da napredne strukture kao što su obeleženi na početak i FIFO postižu dosta bolje rezultate od generičkog algoritma za sve moguće ulaze. Treba napomenuti i da vreme izvršavanja algoritama zavisi direktno od broja čvorova koji se nalaze u grafu. Korelacija između gustine grafa i vremena izvršavanja algoritama nije pronađena. Najzad, zbog značaja transportnih mreža potrebno je nalaziti nove mogućnosti za napredak u ovom polju – bilo nalaženjem novih algoritama bilo optimizacijom postojećih.

Literatura

[1] Andrew Goldberg, Eva Tardos, and Robert Tarjan. *Network Flow Algorithms*. Princeton University, Department of Computer Science, 1990.

[2] Vesna Marinković, Filip Marić, Strahinja Stanojević, and Sana Stojanović-Đurđević. *Konstrukcija i analiza algoritama*. Matematički fakultet, Univerzitet u Beogradu, 2019.

[3] Sargam Monga. Overview of Maximum Flow Problem. on-line at: https://iq.opengenus.org/maximum-flow-problem-overview/.

[4] Sargam Monga. Push Relabel Algorithm. on-line at: https://iq.opengenus.org/push-relabel-algorithm/.

[5] Ferozuddin Riaz and Khidir Ali. Applications of graph theory in computer science. *Third International Conference on Computational Intelligence, Communication Systems and Networks*, 2011.

[6] Univerzitet u Kaliforniji. Applications of Network Flow. on-line at: http://www.cs.ucf.edu/~dmarino/progcontests/modules/netflow2/NetFlow-Apps.pdf.